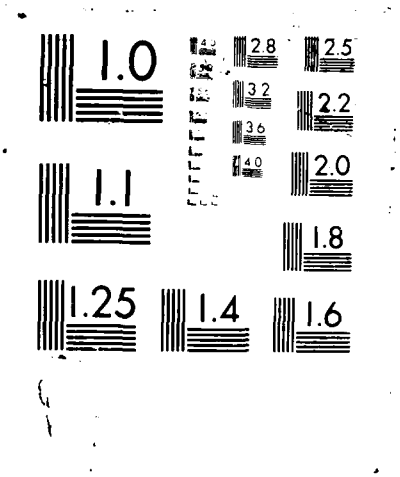
AD-A193 296 NEAR-OPTIMAL SPEEDUP OF GRAPHICS ALGORITHMS USING MULTIGAUGE PARALLEL COMPUTERS(U) WASHINGTON UNIV SEATTLE DEPT OF COMPUTER SCIENCE T D DEROSE ET AL. 1/1

UNCLASSIFIED DEC 86 N00014-85-K-0328 F/G 12/5 NL


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: none	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): Near-Optimal Speedup of Graphics Algorithms Using Multigauge Parallel Computers		5. TYPE OF REPORT & PERIOD COVERED: Technical Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Tony D. DeRose, Lawrence Snyder, Chyan Yang		8. CONTRACT OR GRANT NUMBER(s): N00014-85-K-0328
9. PERFORMING ORGANIZATION NAME AND ADDRESS: University of Washington, Department of Computer Science, Seattle, Washington 98195		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research, Information Systems Program, Arlington, VA 22227		12. REPORT DATE: Dec. 1986 13. NUMBER OF PAGES: 6
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report): Unclassified 15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this report is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Multigauge parallel computers, Quarter Horse microprocessor: graphics algorithm

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A multigauge computer can have its datapath split into independent datapaths that execute operations on "small" data concurrently. In this paper we explain how certain forms of multigauge processing can be implemented with little cost in hardware; we illustrate this "low cost" implementation by describing the multigauging of the Quarter Horse microprocessor, and we report on two practical applications from graphics for which 95% of the theoretically possible speedup is achieved with this low cost implementation. We conclude that because multigauging can be used with other architectural structures it is an effective way of exploiting parallelism.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Near-Optimal Speedup of Graphics Algorithms Using Multigauge Parallel Computers[1]

Tony D. DeRose, Lawrence Snyder, Chyan Yang
Department of Computer Science
University of Washington

Abstract

A multigauge computer can have its datapath split into independent datapaths that execute operations on "small" data concurrently. In this paper we explain how certain forms of multigauge processing can be implemented with little cost in hardware; we illustrate this "low cost" implementation by describing the multigauging of the Quarter Horse microprocessor; and we report on the practical applications from graphics for which 95% of the theoretically possible speedup is achieved with this low cost implementation. We conclude that because multigauging can be used with other architectural structures it is a very low cost way of exploiting parallelism.

1 Introduction

It is not sufficient for parallel computers to speed up computations simply by applying p processors to a problem: High performance can be realized only by incorporating parallelism throughout the design of the machine. Multigauging is a general method of incorporating parallelism into an architecture which is applicable not only to the processor elements of parallel computers but to serial processors as well. In a multigauge machine a B-bit wide data path is designed to be split up into k independent b-bit wide data paths ($B \geq kb$) capable of executing concurrently when the data values being processed are "small", i.e. no more than b bits wide [7, 8]. For example, a standard B=32 bit computer might be partitioned into k=4 separate data paths to concurrently process image data composed of 8-bit pixels. A multigauge machine can be designed to execute in either SIMD mode (the case considered here) or in MIMD mode [7], and the conversion between wide gauge (B-bit width) and narrow gauge (b-bit width) is performed under programmer control.

[1]This paper is supported in part by ONR contract N00014-85-K-0326, DARPA contract MDA907-85-K-0072 and NSF grant DMC-8602141.

Concurrently executing several data paths speeds up a computation through parallelism, but why bother dividing a single data path into pieces? The reason is that some of the most time consuming algorithms must perform within the same computation both brute force processing on "small" data, and sophisticated calculations requiring full precision. Speeding these computations up is likely to pay significant dividends, but it cannot be done simply by adding new instructions to a standard processor [7]. Coprocessors might be developed to handle the "small" data processing, but this presents several drawbacks: There is essentially a doubling of hardware costs, there is bus contention if the processor and coprocessor share the same bus, and if they do not share the same bus then the data has to be moved from one to the other, thus adding unnecessary communications costs; moreover, there is often no high level processing to do until the low level processing is finished, so the coprocessor solution doesn't necessarily add any extra parallelism. In contrast, multigauging requires negligible hardware additions as described below, and so has essentially the same hardware costs as a standard B-bit machine; furthermore, the data does not have to be moved – it is processed where it is. So multigauging is in a sense the "right way" to organize a processor to get extra parallelism at no extra cost. (See [7] for a review of related ideas and research.)

The key question is: Are there important and interesting classes of problems that can use the parallelism of multigauging? The answer is an emphatic "yes". To support this claim, we describe applications, taken from graphics, in which the theoretically maximal speedup of k is nearly achieved. For example:

- For Bézier curve evaluation and display, we report the speedup of 1.94 with k=2 multigauging.
- In the transformation and scan-conversion of line segments, we report speedups of 1.70, 1.99 and 1.9995 for processing 1, 50 and 1000 line segments with k=2.

These speedups demonstrate that practical algorithms can benefit from multigauging, but the examples are hardly unique. All they depend upon is that a substantial amount of the computation involve "small" data values; in graphics, screen coordinates typically require only 10 bits of precision. Other notable application areas include image processing and database processing where there is a massive amount of 8-bit data processing coupled with high level computation. Both problem domains could benefit from multigauging.

Finally, we describe a new concept of *virtual register* which was motivated by the scan-conversion algorithm. This concept, described in a subsequent section, relaxes a tight constraint of SIMD processing, making it more like MIMD processing. This concept might be applicable to other SIMD architectures.

2 Preliminaries

The only information about multigauging that must be added to what has already been said in the Introduction is that we typically limit the narrow gauge to only a single value of k. This restriction is justified because it reduces complexity and hardware – though certain combinations are almost as efficient[9] – and since multigauging may often have a known target application that dictates a suitable value of k, the limitation does not seem serious.

Our approach is to take a 32-bit microprocessor, the Quarter Horse [6], and "convert" it into a multigauge machine. The word "convert" is in quotation marks because we have not fabricated the multigauge Quarter Horse, but we have analyzed each component of the machine to see the impact of supporting multigauging. We begin this section therefore, by explaining the Quarter Horse in sufficient detail to understand its conversion to multigauging, and then continue with a description of the experimental methodology.

2.1 The Multigauging of the Quarter Horse

The Quarter Horse is a 32-bit microprocessor implemented as a single CMOS chip in 90 days as an experiment in fast prototyping [6]. The machine has a typical reduced instruction set with load/store memory reference, a 32 bit instruction, 32 general purpose registers, a dual bus "Mead-Conway" ALU, a barrel shifter, a 32 bit address, and a PLA controller that implements the "typical" instruction in 6 microcycles. The machine uses a 75ns clock.

To convert the Quarter Horse into a multigauge processor, there are three major areas of interest: dividing the processor into k units, memory interface, and control. Although the methodology for dividing the processor into units is the same for all $k < B$, we only consider the $k = 2$ case here [9]. The general purpose registers are divided in half trivially; indeed, if it were not for the virtual registers (see below) there would be nothing to do for the SIMD case since the datapath bus lines are bitwise independent. The primary

impact on the ALU is the addition of carry-in logic. Additional flag bits must be added for MIMD multigauging, but in SIMD mode only one machine can affect the flow of control and we force it to be the "upper gauge," which simply uses the same flag logic the wide gauge uses. Splitting the shifter is complex – using half a shifter uses a quarter of the logic – but the details have little affect on this discussion and will be omitted [9]. The program counter need not be split for the SIMD mode since the instruction stream will use the full sized address, as we now explain.

Since there is only a single stream of instructions, there is only a single stream of instruction addresses and so the memory bus need not be split for instructions. For data, the case is different since the narrow gauge machines each need their own data stream, the memory bus must be split into k units. This also means that the address streams will be narrower since each narrow gauge machine is only capable of producing b bits of address to reference data. If nothing special were done the narrow gauge machines would not be able to reference more than a small part of the memory space. So, we add k segment registers inside the memory system, one for each narrow gauge machine, to provide the high order $B - b$ bits of each data reference. Special instructions are added to the controller to load and store these registers, and the compiler generates the program codes to reference the data.

The control of multigauge machines can become complex in general [8]. MIMD multigauging probably requires a controller for each machine at each gauge unless one implements a "multiported microcode". Bit-serial ($b = 1$) probably requires a different kind of control than larger gauges, and it is not difficult to think of ways of using different control logic for different gauges even in SIMD computation. But we resist this temptation and use a single controller for both wide gauge and narrow gauge; that is, all but a few instructions, for example "fork", have the same meaning in both gauges. This reduces the impact on the hardware substantially and is worth whatever small limitation it imposes.

2.2 Experimental Methodology

We now return to describing the experimental methodology and its assumptions.

Problems are solved using multigauge architectures by writing standard sequential programs. The only difference is that portions of the program run serially in wide gauge (B-bit data), while other portions run in parallel in narrow gauge ($b = B/k$-bit data). Transitions between wide and narrow

gauge are accomplished by augmenting the instruction stream with primitive instructions that implement "fork into k data paths" and "join to become 1 data path." Using the Quarter Horse as a target machine for measurement, we write a program in C and compile it into assembler code [4]. Translating the code into Quarter Horse microinstruction cycles, we obtain a dynamic ratio of the portions run in different gauges. That is, if the number of machine cycles running in wide gauge is T_W and the machine cycles running in narrow gauge is T_N, then the wide gauge fraction of the program, f, is defined as $T_W/(T_W + T_N)$. Based on the measures we compute a speedup coefficient η as $1/[f + (1-f)/k]$.

The best way to explain our methodology is through a specific example. In section 3, we consider the problem of computing points on a Bézier curve with $k = 2$. We compute that the total machine cycles required for the wide gauge mode computation is $270 + N \times 123$, where N is the number of points of evaluation. These wide gauge machine cycles are used for setting up the control points and passing them to the narrow gauge machines. When measured dynamically, the codes which could be run by narrow gauge machines need $N \times 3985$ cycles. Therefore, the wide gauge fraction f in this case is

$$\frac{(123N + 270)}{[(3985 + 123)N + 270].}$$

In another words, if we limit the number of evaluation points to the range [64, ∞] then this ratio, f, would be in the interval [0.0299, 0.0309]. That is, the values of the speedup, η, are in the range [1.940, 1.942]. A closer examination reveals the speedup is not sensitive to N, the number of evaluation points, but is heavily depend on the dynamic ratio between the portions of codes executed in different gauges.

In fact, we apply the Amdalh's Law [1] to measure the upper bound of the speedup. According to Amdalh's Law, if a fraction f of a computation is executable in wide gauge mode then the speedup is bounded above by $(f + \frac{1-f}{k})^{-1}$ given that we have k narrow gauge machines running concurrently. Although this measure is crude, it is a good way to analyze specific problem instances and clearly shows a speedup as k grows and f diminishes.

Note that, as a first order approximation, we assume the machine cycles of "fork" and "join" instructions will not significantly affect the measurement. One may wonder what would happen if we have to include these gauge-changing cycles for a more accurate measurement. This time for the gauge shifting costs about the same number of machine cycles as that of a branch

instruction, i.e., 3 machine cycles. Thus we charge 6 cycles for fork and join. With this assumption, the fraction f is now:

$$\frac{(123+6)N+270}{(3985+123+6)N+270}$$

Following the same computation procedures, we get the speedup, η, in the range of [1.937, 1.939]. With this analysis, less than 1% (=1.94 − 1.93) loss in speed seems insignificant. Having shown the negligible effect of gauge-changing, we do not include this overhead in the following presentation since not all applications need significant portions of gauge-changing activities like the Bézier evaluations.

3 Evaluation of Bézier Curves

As our first example of the use of multigauging in interactive graphics, consider the problem of evaluating and displaying a Bézier curve [2]. A Bézier curve $Q(u)$ of degree n is of the form

$$Q(u) = \sum_{i=0}^{n} V_i\, B_i^n(u), \qquad u \in [0,1], \tag{1}$$

where $V_0, ..., V_n$ are controlling points commonly called control vertices, and $B_0^n(u), ..., B_n^n(u)$ are the n^{th} degree Bézier blending functions defined by

$$B_i^n(u) = \binom{n}{i} u^i (1-u)^{n-i}.$$

These curves are often used in interactive design systems by having the user specify the control vertices, and having the system compute and display the resulting curve by repeatedly evaluating $Q(u)$ for many different values of u. These computations are typically done using floating point arithmetic. However, by using the evaluation algorithm of de Casteljau [3], and by assuming that the display is a raster of $2^c \times 2^c$ pixels, the evaluation for a fixed value of u can be done using only integer arithmetic of slightly more than c bits precision, as we now show:

The computation (1) can be easily shown to be identical to (2) for a fixed value of u, with $Q(u) = V_0^n$ and $V_i^0 = V_i$.

$$V_i^j = uV_i^{j-1} + (1-u)V_{i+1}^{j-1}, \qquad i,j = 0,1,...,n \tag{2}.$$

We can replace parameter u by $t = 2^r u$, i.e., we scale up the parameter. The computation (2) now becomes

$$V_i^j = \frac{1}{2^r}(tV_i^{j-1} + 2^r V_{i+1}^{j-1} - tV_{i+1}^{j-1}), \qquad t \in [0, 2^r] \qquad (3).$$

Therefore, only integer arithmetic is required provided only integer values of t are chosen. More specifically, if $c = 10$ and up to 64 ($r = 6$) points of evaluation are required per curve segment, then the computation can be done to 10 bits precision using 16-bit integer arithmetic. This computation can therefore be performed in two narrow gauge streams if the wide gauge width B=32 bits. In most cases, a precision of 64 points per curve segment is sufficient to give an accurate understanding of a curve's shape. The computation could be described as follows:

1. Initialize the control points with the wide gauge;
2. /* outer loop */
   The wide gauge machine invokes N = 2^r points of the Bézier evaluation, i.e., it iterate step 3;
3. /* inner loop */
   Equation (3) is evaluated for a given integer parameter t by the narrow gauge machines.

To get an estimate of the efficiency of the multigauge version of Bézier evaluation, we compiled the Bézier computation program (written in C) into assembler codes[4] of the Quarter Horse. We then got a dynamic ratio of wide gauge and narrow gauge codes by translating the instructions into Quarter Horse microcycles. For cubic Bézier evaluation ($n = 3$), the ratio of machine microcycles required for wide gauge vs. narrow gauge computation is 3.09:96.9, and for the quartic case ($n = 4$) the ratio is 2.03:97.97.

Since the narrow gauge computations are done in parallel on two machines, the number of microcycles they require is cut in half. If T is the time required by a wide gauge machine, the speedup coefficient is

$$\eta \cong \frac{T}{0.03T + 0.485T} = 1.94,$$

which nearly reaches the optimal value of 2. In the cubic case when $k = 3$ or 4, the value of η is 2.83 or 3.67, respectively. For quartic Bézier curves, η is 1.96, 2.88, or 3.77, for $k = 2, 3$, or 4, respectively. In each case, the speedup is nearly optimal, indicating that multigauging is indeed appropriate for narrow data width, compute-bound applications.

4 Scan Conversion

To demonstrate the use of multigauging to trade off accuracy for speed, consider a problem common to many graphics applications — the processing of a set of line segments through a transformation and scan-conversion display pipeline. In our example we assume that:

1. The line segments are specified by the coordinates of their endpoints relative to an arbitrary coordinate system;
2. The display is a raster of $2^c \times 2^c$ pixels;
3. Lines are processed through the pipeline by transforming the endpoints into device coordinates using a 4×4 homogeneous matrix, followed by scan-conversion into the display.

Multigauging is used to solve this problem as follows. The transformation matrix is constructed in wide gauge using 32 bit arithmetic. It is then scaled and truncated to 10-bit precision. Since the matrix is homogeneous, scaling each entry does not change the transformation. The 10-bit approximation to the transformation matrix is then fed to the narrow gauge machines. Having transformed the endpoints to its new coordinates, we now ready to fill in the pixels between them.

Given a pair consisting of a starting point $P_1(x1, y1)$ and an ending point $P_2(x2, y2)$, Bresenham's algorithm draws a line from P_1 to P_2. The algorithm is attractive in practice because it uses only integer arithmetic. To simplify our discussion, we give a brief description of the algorithm assuming the slope of the line segment is between 0 and 1. Suppose that we are now drawing from the left to right at the pixel (x, y) then the next point should be either $(x+1,\ y)$ or $(x+1,\ y+1)$ depending on a decision variable d. The decision variable d is a measure indicating which of the two points is closer to the true line. Note that if we want to "march" from the opposite direction, then the X-value and the Y-value should be decremented since d is independent of the marching direction. We use the following computations for the scan conversion measurement:

1. Compute sine and cosine values in wide gauge, scale up by 2^{10}, round off to integers;
2. Using narrow gauge calculations, transform endpoints in parallel and scale back down by 2^{10}. The floating point numbers are now converted into integers.

3. Compute Bresenham's algorithm in narrow gauge mode from opposite directions in parallel.

The Bresenham's algorithm is described as follows:

1. Compute some housekeeping constants such as incr1, incr2, and decision variable d;

2. /* Decide which one, P1 or P2, as a starting point (x, y) depending on either $(x1 > x2)$ or $(x2 > x1)$; treat the other point as an end point $xend$ for the termination test; */

```
if x1 > x2
    then begin
            x := x2;
            y := y2;
            x := x1
            end
    else begin
            x := x1;
            y := y1;
            xend := x2
            end
```

3. /* In the inner loop of Bresenham's computations, the variable sgn indicates the marching direction */

```
output(x, y);
while x < xend do begin
            x := x + sgn;
            if d < 0
            then d := d + incr1
            else begin
                    y := y + sgn;
                    d := d + incr2
                    end
            output(x, y);
end
```

Notice that there are two branching tests, i.e., the **if** statements, impose certain problems of implementing this algorithm in SIMD mode. We over-

come the first **if** by employing a *virtual register* scheme in which a reference to a register address will have a different effect in different narrow gauge machines. By using *virtual registers*, the narrow gauge machines can cooperate to process each line segment in roughly half the time required by a wide gauge machine.

The virtual register scheme places a special programmable switch in front of two working registers such that each narrow gauge machine treats the starting point of the other machine as its ending point. Scan-conversion is done by having the narrow gauge processors "march" from opposite ends of the line segment toward the middle using a variant of Bresenham's algorithm [5]. It is easy to show that in the worst case this parallel algorithm is no more than 1 pixel off the true line, and almost always gives the same result as Bresenham's algorithm. This will become clear momentarily.

We shall find it convenient to introduce a small amount of notation: Let WG represent the 32-bit wide gauge machine, and let NG_1 and NG_2 represent the two 16-bit narrow gauge machines. We store $x1$ and $x2$ into registers, say R1 and R2. Suppose $x1 < x2$. We want NG_1 to see this assignment, (R1 < R2), but we want NG_2 to see the opposite, (R1 > R2). We can achieve this effect at the cost of a few switches inserted between the registers and their addressing lines[9].

In terms of precision, the only time this algorithm is ever one point off Bresenham's algorithm is when a line segment has a slope of 0.5. This is due to the asymmetry of the branching test **if** $d < 0$, and this single pixel shift happens only over half of its line segment because NG_1 and NG_2 are marching in opposite directions. NG_2 draws half of the line segment on pixels with one grid point shifting upward or downward compared with the line segment if drawn by NG_1 from the opposite direction. This imprecision is immaterial in dynamic interactive graphics.

Using an analysis similar to the one for Bézier curves, we have measured speedups of 1.70, 1.99, and 1.9995 for the processing of 1, 50, and 1000 line segments, respectively. Thus, with a minor investment in hardware to implement multigauging and virtual registers, one can expect transformation and scan-conversion pipeline throughput to be significantly improved.

5 Conclusions

Using two simple application examples of practical importance, we have demonstrated that multigauging is a prudent method of increasing system

performance. There are clearly many other applications for which comparable speedups are possible. Throughout the presentation we have also discussed important issues of designing a multigauge machine, and we have demonstrated that multigauging can be incorporated into an existing architecture with few modifications.

Acknowledgements

This work has benefited immensely from the work of the other members of the Quarter Horse design team, Sam Ho, Barry Jinks, Tom Knight, Jim Schaad and Akhilesh Tyagi and from the technical staff of the Northwest Laboratory for Integrated Systems. In addition, the Quarter Horse compiler written by Kay Crowley has been of great help to our experimental work.

References

1. G. M. Amdalh, "Validity of the Single Processor Approach to Achieving Large-Scale Computing capabilities," *Proc. AFIPS, Vol 30*, 1967, pp.483-485.

2. B. A. Barsky, "A Description and Evaluation of Various 3-D Models," *IEEE CG&A* Jan. 1984, pp. 38-52.

3. P. de Casteljau, "Courbes et surfaces à poles," André Citroën Automobiles SA, Paris.

4. K. E. Crowley, *Using a Retargetable Compiler to Evaluate the RISC Architecture*, Master Thesis, Department of Computer Science, University of Washington, June, 1986.

5. J. D. Foley and A. Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, 1982.

6. S. Ho, B. Jinks, T. Knight, J. Schaad, L. Snyder, A. Tyagi, and C. Yang, "The Quarter Horse: A Case Study in Rapid Prototyping of a 32-bit Microprocessor Chip," *IEEE ICCD: VLSI* 1985, pp. 161-166.

7. L. Snyder, "An Inquiry into the benefits of Multigauge Parallel Computation," *IEEE ICPP 1985*, pp. 488-492.

8. L. Snyder and C. Yang, *An Investigation into the Design Costs of a Single Chip Multigauge Machine*, Technical Report, TR86-06-01, Department of Computer Science, University of Washington, June, 1986.

9. C. Yang, *An Investigation of Multigauge Architectures*, Ph. D. Dissertation, Department of Computer Science, University of Washington, in preparation.

END

DATE

FILMED

DTIC

JULY 88